A Diversity Model Based on Failure Distribution and Its Application in Safety Cases

L. Chen, J. May[1]

***Abstract*— This work develops a new basis for evaluating the reliability benefits of diverse software, based on fault injection testing. In particular, the work investigates new forms of argumentation that could in principle be used to justify diversity as a basis for the construction of safety claims. Failure distributions of two versions of diverse software under various fault conditions are revealed separately by fault injection methods, and then the common failure probability of the version-pair can be estimated. The approach is justified theoretically, and cross validated with other work. This method is also used to explain the fundamental influence of failure distributions on diversity. Furthermore, the unique capabilities of the method are demonstrated by implementation of the fault injection test on a program pair**

***Index Terms*— Software Diversity, Safety Case, Safety Critical System, Fault Injection, Multi-version, Reliability.**

Abbreviations:

AFR	Average failure rate
CFP	Common failure probability
EL	Eckhardt and Lee
FD	Failure distribution
FI	Fault injection
IFP	*s*-Independent failures model
LM	Littlewood and Miller
MR	Meulen and Revilla
MSF	Mmodified sign function
NVP	New version-pair
RI	Reliability improvement
SFP	Single failure probabilities
SIL	Safety integrity level

Notation:

P	Probability of system/software failure
R	Reliability of system/software
Ω	Finite input space
ω_i	*i*th input point
$\theta(\omega_i)$	Modified sign function on input ω_i
X, x	A random, given input
Π, π	A random, given program

L. Chen is with the Safety Systems Research Centre, University of Bristol, Bristol, BS8 1TR, UK (e-mail: L.Chen@bristol.ac.uk)
J. May is with the Safety Systems Research Centre, University of Bristol, Bristol, BS8 1TR, UK (e-mail: J.May@bristol.ac.uk)

F, f	A random, specific faulty condition
$U(X = x)$	Operational profile of a given input x
$S(\Pi = \pi)$	Probability of selecting a particular version π
$H(F = f)$	Occurrence probability of a specific faulty condition f
$\nu(f, x)$	Modified sign function
$\varphi(\pi, x)$	Score function

I. INTRODUCTION

The need for research into models of diverse failure behavior can be explained by reference to redundant channel software-based architectures in critical systems. Use of multiple redundant trains of equipment is a key method for achieving reliability in critical systems built from hardware. The presence of common software in such trains introduces possibilities for common cause failure which defeat this approach. Thus software diversity has the potential to play a central role in the design of defences against failures in safety critical systems built with redundant trains, by reducing the scope for common cause failures. Diversity design aims to build multiple different versions of a program fulfilling the same requirement specifications, in the hope that any remaining faults in the versions exhibit different failures, in terms of both failing on different inputs or failing differently on common inputs [1]. This is possible in principle since a common requirement can be implemented in different ways, e.g. using different functions, structures or languages etc., leading to differences in the remaining faults and failure patterns. However, diversity measurement remains an open topic i.e. the level of reliability gain resulting from the use of software diversity is unknown. An immediate practical consequence of this is that it is difficult for safety cases to claim any formal benefits resulting from the use of diversity in multiversion safety critical systems. A corollary is that it is not clear how to design diverse software for reducing the possibility of common failures, other than to use diverse development processes and make the software versions 'different' from each other.

Given the fact that current development methods cannot ensure fault-free software, fault tolerant techniques are used extensively in practise. These are based on diverse behavior directly (N-version programs) or indirectly (e.g. extra diagnostic or recovery functions), and are especially important in safety-critical systems [1]. Applications of software diversity can be found in critical flight-controllers [2], railway signaling and control systems [3], and nuclear reactor protection systems [4]. Safety standards usually require a form of warranty to be made for any application of software built for use in safety-related applications such as those used in the nuclear, medicine and aerospace sectors [5, 6]. Ideally, the arguments in the safety cases constructed to satisfy this, would demonstrate the high reliability required for these systems in a quantitative scientific way. In fact, the state of the art is that the diversity design can be used only as a qualitatively beneficial defence for common cause failures. The associated safety cases can not quantify the reliability enhancement achieved because there is no theoretical model to assess the reliability implications of diversity between two or more particular program versions. One major reason for this is the low chance of observing real failure behaviours in high reliability safety-critical systems, so there is insufficient empirical evidence to arrive at any conclusion regarding guidelines for design and assessment, or support any measurement-based estimation models.

One approach to this problem uses fault injection (FI) to evaluate diverse version behaviours [7, 8]. Such an approach attempts to establish the relationship between faults and failure behaviours and the reliability characteristics of diverse software versions [9]. Fault simulation has been developed to evaluate software by deliberately injecting faults into a system or a copy (such as a prototype or a simulation), to observe the effects of these faults on the system behaviour [10, 11]. Various software fault injection techniques have been employed [7, 12,13].

The difficult problem of diversity assessment has been the subject of a significant body of empirical research. The experiments in [14,15] indicated that it would be unreasonable to claim that diverse software versions fail independently. Further work investigated the factors influencing the independence of diverse developments e.g. the use of different programming languages [16] and the setup of communication protocols between different developers [17,18].

Probabilistic models do exist to describe general properties of diversity over populations of programs, such as those of Eckhardt and Lee (EL) [19], and Littlewood and Miller (LM) [20], but these models do not in general yield diversity measures for N-version software, or depend upon parameters that are very difficult to estimate. Recent work has estimated a conservative bound for the failure probability of a program-pair in a special case [36,37], but a general solutions are not available The EL and LM models attempt to describe what is true 'on average' for a large population of programs. Subsequently, an online collection of programs has been used for demonstrating the LM model [21]. In Meulen and Revilla's (MR) experiments [22], thousands of programs have been gathered for one common specification. Although the program structure is too simple, and the number of developed versions far too great, to represent any kind of realistic practice, some principles for diversity phenomena have been observed and discussed.

This paper aims to formulate a failure distribution (FD) model to reveal the diversity between two particular software versions in a more formal way than is currently possible. The FD model is a development of the fault injection approach to estimate software

diversity based on a statistical distribution of software failures over the input space. It is being developed as a theoretical framework to support the eventual use of fault injection for diversity claims within future system safety cases. The approach of the FD-model is cross validated with both the LM-model and the MR-experiment in this paper. The paper also identifies various key factors that influence safety claims inferred from failure-distribution experiments. It also explains how the range of possible failure distributions caused by faults appear to be circumscribed, and why this may be an important feature in the analysis of software diversity. Finally, a fault injection test on a version pair demonstrates that the FD model can estimate the diversity of a particular 1-out-of-2 system, and why this differs from the general conclusion drawn from testing a large population of programs based on the EL/LM model.

II. FAILURE DISTRIBUTION MODEL

A. Diversity Quantification

The concept of diversity in software engineering has been conceived to evaluate the benefits of software constructed by multiple functional branches all designed to achieve the same task. These branches are called components, divisions, versions or channels in different applications. They are expected to show diverse failure behaviours. Then a voting mechanism compares the outputs from the versions to reduce the risk of failures affecting system level operation. There is still no well-recognised quantitative index capturing diversity directly as a system property. The effect of diversity is commonly characterised through a comparative reliability improvement. Software Reliability is generally defined as the probability of failure-free software operation for a specified period of time in a specified environment [23]. Denoting the failure probability and reliability of software as P and R respectively, $R = 1 - P$ follows and is used throughout this paper.

For a multiple version software system, the common failure probability (CFP) between versions (or channels) is the key parameter required to calculate the system reliability. For example, in some situations detection of a dangerous condition by 1-out-of-2 channels within a 2 channel protection system is deemed sufficient to shut down an industrial process. Hence system failure under this situation is the inability of both channels to recognise a dangerous plant condition (hence the '1-out-of-2' system definition), and the CFP gives the system reliability R directly.

The long-standing issue in diversity quantification is that the CFP of multiple versions can not be simply deduced directly from the single failure probabilities (SFP) of each version. This paper studies 1-out-of-2 systems, but the conclusion can be extended to general 1-out-of-n voting systems. For a 1-out-of-2 system with versions A and B, the issue is to estimate CFP P_{AB} from SFPs P_A and P_B of the version pair respectively.

One way to proceed with a 1-out-of-2 system would be to assume the statistically independent failures model (IFP) for the CFP:

$$P_{AB} = P_A P_B \qquad (0)$$

on the grounds that two versions were 'developed independently'. However, the IFP often results in an unsafe (optimistic) estimate. CFP is decided by not only the failure rates of two versions/channels individually, but also by the correlation of the failure behaviours [19]. Approaches to elucidate failure correlation explore how and why the versions may fail commonly on some points of the same input space with a greater than average probability. This issue was identified in the early studies of diversity [1,25].

B. CFP definition

The definition of failure probability P in this paper is based on the concept of software input space. Assuming software developed to a requirement specification has a finite input space Ω which contains N_ω input points ω_i (i is from 1 to N_ω), and the behaviour of software acting on an input ω_i is quantified by defining a modified sign function (MSF) $\theta(\omega_i)$:

$$\theta(\omega_i) = \begin{cases} 0 & \text{success on } \omega_i \\ 1 & \text{failure on } \omega_i \end{cases} \qquad (1)$$

Then the software failure probability P is:

$$P = \sum_{i=1}^{N\omega} \theta(\omega_i) U(\omega_i) \qquad (2)$$

Where $U(\omega_i)$ is operational/user profile: the probability that an input ω_i is selected at a random point in software operation [26].

The two versions of a program A and B are developed according to the common requirement specification, which means both versions also have the same input space Ω. Similarly defining a sign function and failure probability P_A and P_B for both A and B:

$$P_A = \sum_{i=1}^{N\omega} \theta_A(\omega_i) U(\omega_i) \qquad (3)$$

$$P_B = \sum_{i=1}^{N\omega} \theta_B(\omega_i) U(\omega_i) \qquad (4)$$

The common failure probability P_{AB} for version A and B is:

$$P_{AB} = \sum_{i=1}^{N_\omega} \theta_A(\omega_i)\theta_B(\omega_i) U(\omega_i) \qquad (5)$$

Comparing (3), (4) and (5), it is easy to see that the IFP $P_{AB} = P_A P_B$ does not necessarily apply to a particular version-pair, even if the version pair has been created 'independently', e.g. with different teams, source code and languages etc. The IFP might intuitively be expected to describe the behavior of a random version pair from a population containing all possible pairs. However, empirical evidence showed that a CFP obeying the IFP is unrealistic in practical applications [1]. A plausible speculation is that there is some amount of deterministic control over the failure domains in software development, so for a particular program the failure distribution over the input space is not fully known, but it is not statistically random: e.g. the basic functional tests used during software development ensure that all versions share some known common success areas in the input space.

Fig. 1 sketches (Venn diagram) three special failure distributions over an input space of rectangle area Ω illustrating three CFPs: $P_{AB} = 0$, $P_{AB} = P_B$ and $P_{AB} = P_A P_B$, where the shadow areas A, B and AB indicate the corresponding failure regions of the single version A, B, and the version pair A and B.

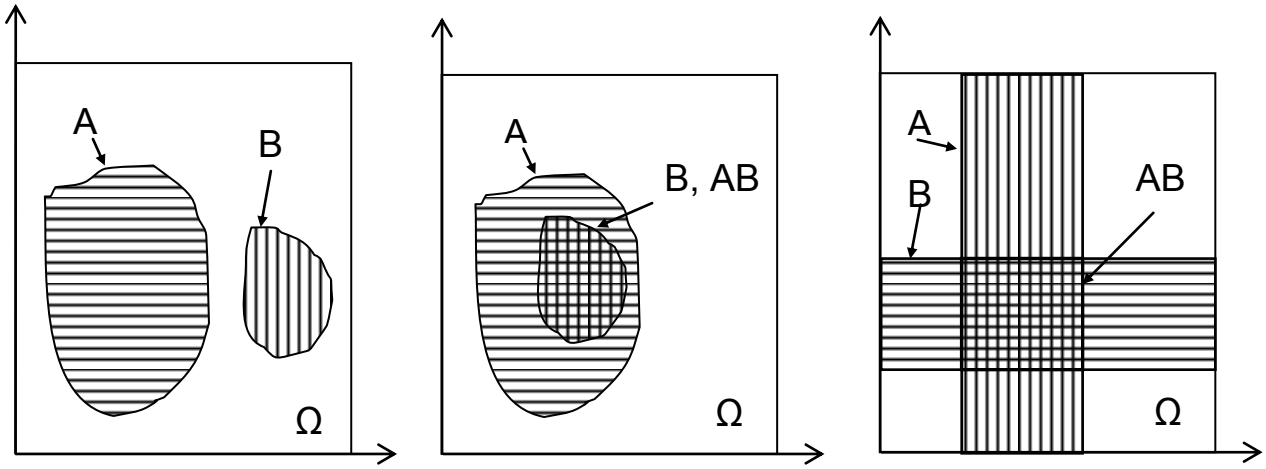


Fig. 1 Three special situations for common failure probability

The three cases represent:

- no-overlapping: maximum diversity,
- totally-overlapping: no diversity,
- orthogonal failure domains between version A and B.

The 3rd case presents a typical and simple situation that the diversity satisfies the IFP model: when the two failure regions *A* and *B* are orthogonal to each other on the plane, numerically there is:

$P_A = \dfrac{area\ of\ failure\ region\ A}{area\ of\ input\ space\ \Omega}$,

$P_B = \dfrac{area\ of\ failure\ region\ B}{area\ of\ input\ space\ \Omega}$ and

$$P_{AB} = \frac{area\ of\ failure\ region\ AB}{area\ of\ input\ space\ \Omega} = P_A P_B$$

The orthogonal condition being sufficient (but not necessary) for two variables to be independent, this, and its relationship to zero correlation, have been explained in many studies [27].

C. Previous diversity models

For equation (5) to be useful in determination of CFP, it requires knowledge of the software behaviours on every input. Exhaustive test is an impossible task for most systems, so development of practical diversity evaluation models must use estimation of $\theta(\omega_i)$ rather than complete test results.

Two well-known diversity models use $\theta(\omega_i)$ to examine version correlation: the EL model and the LM model. In the EL model, the randomness present in the development process was modelled as a random selection of a program Π from the set of N_Π possible program versions that 'could' be developed to the same requirement specification, and the probability of selecting a particular version π was denoted as $S(\Pi = \pi)$, where the capital letter and small letter are used to distinguish a random variable and its given value. The operational or usage (having same meaning in this paper) profile of a random input X to any program can still be described by $U(X = x), x \in \Omega$. Then the average probability of a program version failing on a given input x is defined as a "difficulty function" $\theta(x)$ since it reflects a property of x averaged across the population of possible program versions:

$$\theta(x) = \sum_{\pi} \varphi(\pi, x) S(\Pi = \pi) \qquad (6)$$

In (6), $\varphi(\pi, x)$ is defined as a "score function" and is similar to $\theta(\omega_i)$ in (1): it has value zero when a program version π succeeds on a given input x, and otherwise value one. Moreover, $\theta(x)$ is regarded as an extended version of the original sign function $\theta(\omega_i)$ (which takes a deterministic value 1 or 0), taking probability values within a continuous range [0.1]. Then SFPs of A and B, regarded as two randomly chosen versions from one population can both be written as:

$$P(\Pi \text{ fails on X}) = \sum_{x} \theta(x) U(X = x) \qquad (7)$$

The equation (7) describes the probability of a randomly chosen program failure on a random input point. The CFP of a 1-out-of-2 system formed from programs Π_A and Π_B therefore is:

$$\begin{aligned} &P(\text{Both } \Pi_A \text{ and } \Pi_B \text{ fails on X}) = \sum_{x} \theta(x)\theta(x) U(X = x) \\ &= P(\Pi_A \text{ fails on X}) P(\Pi_B \text{ fails on X}) + \mathrm{cov}_x(\theta(X), \theta(X)) \end{aligned} \qquad (8)$$

where $\mathrm{cov}_x()$ is standard covariance function that defines the version correlation. The $\mathrm{cov}_x(\theta(X), \theta(X))$ is greater than or equal to zero, and 'equal' if and only if $\theta(x)$ is uniform for all x. Hence in the EL formulation IFP is always an optimistic estimation of the CFP. The EL model explains this as the phenomenon that similar difficulties exist in the implementation of some parts of the requirement specification, irrespective of the program chosen, so the chances of errors in the two versions built by different developers are correlated across the input space.

The LM model argued that the two random versions Π_A and Π_B may be simulated as separate selections from two different program populations. Then based on (4), there are two different difficulty functions for $\theta_A(x)$ and $\theta_B(x)$ over the same input space. Similarly, there are two different SFPs, $P(\Pi_A \text{ fails on X})$ and $P(\Pi_B \text{ fails on X})$. Correspondingly, a CFP for the 1-out-of 2 system by Π_A and Π_B based on LM model is:

$$\begin{aligned} &P(\text{Both } \Pi_A \text{ and } \Pi_B \text{ fails on X}) = \sum_{x} \theta_A(x)\theta_B(x) U(X = x) \\ &= P(\Pi_A \text{ fails on X}) P(\Pi_B \text{ fails on X}) + \mathrm{cov}_x(\theta_A(X), \theta_B(X)) \end{aligned} \qquad (9)$$

In the LM model (9), the correlation term $\mathrm{cov}_x(\theta_A(X), \theta_B(X))$ can be either positive or negative, and there is still a possibility that the IFP applies for some version-pairs.

The analytic approach of the EL and LM models successfully explains some practical phenomena and theoretical features of software diversity, but they do not yield a general method of diversity estimation for real programs. The problem is that the only figure which might be considered to indicate the CFP of a particular program pair is the average CFP of the program population which the version-pair were randomly selected from, i.e., the difficulty function $\theta_A(x)$ or $\theta_B(x)$ was estimated by its average value over the population. For example, at each point x in the input space, if the probability of selecting a particular version π was uniform, $\theta(x)$ in (6) is simply the number of failed versions N_{failed} divided by the total number of program versions in the population:

$$\theta(x) = \frac{\sum_{\pi} \varphi(\pi, x)}{N_{\Pi}} = \frac{N_{failed}}{N_{\Pi}} \qquad (10)$$

D. Failure Distribution (FD) Model

The EL/LM models provided a theoretical way to describe the statistical relationship between random factors in the development process and the CFP. Based on the selection of a version-pair from one or two large populations of programs, the necessity to look beyond the assumption of independence was clarified. But for real applications, especially in the case of estimation for present safety-critical systems based on inherent computer infrastructures and operation modes, it is impossible to generate the program population, both in principle (because there is no mechanism for deciding which programs are in the population and which excluded) and in practise due to the effort that would be required to build a large enough population. Furthermore, the concept that any such developments correspond to random selection of a pair from a program pool or pools is not justified.

The factors driving CFP include three main types:

- fault planting

Fault creation occurs at every stage and in every aspect of the development process, e.g., in requirements, design or coding, and due to tool bugs or human errors;

- failure triggering

When software contains faults, the failure occurrences can still be random because specific conditions, e.g. certain system configurations or input sequences, are required to trigger a fault, and there is randomness in condition occurrence.

- fault to failure propagation

This is a structural feature, in the sense that the software architecture decides the propagation route from a fault to corresponding failures, which in turn determines the locations of failures in its input space.

Statistical modelling can in principle describe the random factors analytically, and parameterizing and refining these factors will provide improved routes to assessment. For example, the operational profile $U(x)$ has been introduced in various reliability models to handle the relative frequencies of inputs [28]. As another example, the EL and LM models use a random parameter Π to describe the uncertainty in both software structures and residual faults, i.e. through random selection of the version from a program pool. However, for a given version pair, the software architectures are deterministic, which admits a different problem construction. For a particular program, assume that all possible faults form a set of size N_F (that can be infinite without impacting the following analytic results). 'A fault' in the set can be a condition corresponding to a single anomaly or combination of anomaly(ies) in software. Then the uncertainty associated with a program's correctness lies in the likelihood of it containing elements of the fault set. Denote this uncertainty using a variable F representing the program containing a random fault from the set i.e. under a random faulty condition. The occurrence probability of a specific faulty condition f is denoted as $H(F = f)$, which is introduced in a role similar to the operational profile of inputs. Then the average probability $\theta(x)$ of a program version failing on a given input x:

$$\theta(x) = \sum_{f} \nu(f, x) H(F = f) \qquad (11)$$

where $\nu(f, x)$ is a modified sign function similar to the one in (1), and whose range has only two values: 0 when the program with a fault f performs successfully on the input x, otherwise: 1.

An input that triggers a fault to cause a failure can be defined as a failure point under the faulty condition. Then $\theta(x)$ can be defined as a failure distribution (FD) function: ***it indicates the probability that an input x in the input space is a failure point under a random faulty condition.***

Similarly, the failure distribution functions $\theta_A(x)$ and $\theta_B(x)$ for version A and B respectively can be obtained from (11). The failure probability of a particular program π_A is:

$$P_A = P(\pi_A \text{ fails on X}) = \sum_x \theta_A(x) U(X = x) \qquad (12)$$

And for a particular program π_B, there is:

$$P_B = P(\pi_B \text{ fails on X}) = \sum_x \theta_B(x) U(X = x) \qquad (13)$$

The CFP of a particular pair of versions π_A and π_B as a 1-out-of-2 system is:

$$P_{AB} = P(\text{Both } \pi_A \text{ and } \pi_B \text{ fails on X}) = \sum_x \theta_A(x)\theta_B(x) U(X = x) \qquad (14)$$

The FD model (14) describes correlation of fault-failure propagations between two given software architectures, and then calculates the probability of a version-pair failing coincidentally at a random input under a random faulty condition (the fault is from the fault population and occurrence frequency). A potential advantage of the FD model is that a fault population can be constructed in practise, using fault injection testing [29]. The software structures affect diversity because the fault-failure propagation varies between versions. For example, an identical code segment appearing in two versions might contain exactly the same fault (in the sense of incorrect code), but the propagation of the fault to their failures can differ because of the code diversity in the remaining code of the two versions. Fault injection techniques disclose these relations by attempting to simulate all possible faulty conditions. This seems to cause a problem similar to the population of programs in the EL/LM models, since the number of possible faults is infinite. However, the statistical calculation in FD model can be based on identification of failure regions rather than fault sets: and empirical results give some initial indications that the failure patterns in a program's input space may be finite. If the definition of variables F and f is changed to refer to failure patterns, the FD model can be applied (although the frequency of failure occurrence needs to be determined differently).

The difference between the FD model (14) and EL/LM models lies in the construction of populations for the statistical calculations of $\theta(x)$: they are based on a fault set and a set of programs, respectively. So the issue of population representativeness is shifted from one analysis domain (multiple different programs) to another (multiple different faults). Representativeness problems remain, but, in making the shift, these problems can be considered in a framework which is practically achievable (as opposed to the development of a large population of programs built to a common specification, which is clearly not a practical approach). Whilst injecting faults effectively generates a set of programs, it is a more restricted set than that envisaged in the EL/LM conceptual approach; the program set resulting from injection is more specific to the program pair under test. Thus the two approaches reveal different attributes of a system with diversity design. The problem of diversity evaluation in practice involves assessment of two (or more) existing specific program versions that are chosen to operate together. Those versions have properties that should form a central part of the evaluation, and certainly influence the possibilities for and nature of faults in the software. Analysis of a program population built by different people/methods from the specification ignores that specificity - it is analogous to judging the efficacy of a medical drug on an individual by assessing the efficacy of a drug across a sample of people. Numerically, FD and EL/LM models may produce the same results under some special conditions:

- For the FD model, defining the number of faults that cause a version to fail at an input x as N_{failed}, then

$$\theta(x) = N_{failed} / N_F \qquad (15)$$

- If we make the two populations the same (the program pool for the EL/ML model is defined to be the collection of programs formed by injecting faults in two particular versions), then we have $N_F = N_\Pi$, and the equations (10) and (15) give the same numerical result.

III. APPLICATIONS OF THE FD MODEL

A. The relevance of the Failure Distribution (FD) model to Safety Cases

To claim benefits from fault-tolerance or any high reliability designs in safety-critical system applications, a safety case is built [30, 31], and is required by various safety standards including IEC 61508 [32]. The safety case is generally defined as a defensible structured argument, supported by evidence, to justify that a system is acceptably safe in a given context; and normally is broken down into claims about different attributes for the system, e.g.: reliability, availability or security etc. Claims for reliability would normally be supported by statistical arguments. The motivation for developing an empirical fault injection method to assess attributes of multiversion software is clear: little direct statistical evidence is available to describe failure behaviors of high-

reliability software. However, the use of such an approach in safety cases requires theoretical and practical validation. The FD model is a step along that path, providing a theoretical framework for a fault injection approach, which can in principle, be used to estimate diversity for two or more software versions.

Observation of the CFP in a fault injection experiment requires the failure distributions of both versions corresponding to all possible faults with size smaller than the SFPs. In practical testing, faults can be generated and then sieved according to the SFP that they cause. Another question concerns the number of faults to use for simulation. Empirical observations show that, unlike the number of faults, the number of failure patterns in the input space of software can be finite and practically manageable, so exhaustive searching of failure patterns is conjectured to be equivalent to 'all faults' FI stop criterion when using [21].

Based on 1-out-of-2 multiversion systems, the FD model formalizes assessment of CFP for a multiversion system constructed by two versions whose SFPs are known, and raises the possibility of demonstrating how a high reliability system may be constructed from two diverse versions of software with lower reliability. There is an important open question for safety cases, for which the FD model shows promise:

- If a system is constructed from two diverse versions A and B with their SFPs less than a specified small value $\widetilde{P}_A$ and $\widetilde{P}_B$ respectively, what can be expected for its CFP? In the language of safety cases, can a required SIL4 system defined in IEC61508 be constructed by two different SIL3 channels [22]?

The answers, based on the FD model, involve two steps:

- Use of fault injection to find out the failure distribution $\widetilde{\theta}_A(x)$ or $\widetilde{\theta}_B(x)$ for all faults with size smaller than $\widetilde{P}_A$ and $\widetilde{P}_B$ respectively.

- Estimation of CFP based on (14), using $\sum_x \widetilde{\theta}_A(x)\widetilde{\theta}_B(x)U(X=x)$.

B. Diversity under Varied Failure Distributions

Fig. 1 gives a simple but direct visualization that diversity is fundamentally determined by the actual distributions of the failure points in the common input space for two particular versions. Orthogonal failure regions are a special case where IFP is applicable to a 1-out-of-2 system. Based on the EL model that defines a "difficulty" as a probability of an input causing a software failure (an intuitive explanation might be that this is due to software functions handling different inputs needing different 'effort' to implement, and therefore having different chances to contain bugs), the IFP describes two randomly chosen versions only if their difficulty distributions are uniform i.e. a situation where the failures for both versions have same chance to happen randomly anywhere in the input space. But some characteristics of software development can limit the statistical randomness needed for IFP, i.e. the common requirement specification drives the two versions towards similarity of fault-failure propagation relations; and, common functional tests reduce the areas where failures can possibly happen. The FD model can provide tangible explanations of the role of failure distributions in diversity analysis.

Case 1: random failure distribution. Based on the FD model in equation (14), assuming $\theta_A(x)=\alpha_A$ and $\theta_B(x)=\alpha_B$ for all $x\in\Omega$, α_A,α_B are two constants that means uniform distributions, then there is:

$$P_{AB} = P(\text{Both } \pi_A \text{ and } \pi_B \text{ fails on X}) = \sum_x \alpha_A\alpha_B U(X=x)$$

$$= \sum_x \alpha_A U(X=x)\sum_x \alpha_B U(X=x) \qquad (16)$$

$$= P(\pi_A \text{ fails on X})P(\pi_B \text{ fails on X}) = P_A P_B$$

This shows that IFP describes the system under uniform failure distributions, the same conclusion made using the EL model.

Case 2: partly tested input space. A special scenario can be set up to quantify varied failure distributions of two versions having passed some common tests. Assuming the SFPs for versions A and B are $P_A=\alpha_A$ and $P_B=\alpha_B$ as in (16), and the input space is composed of two non-overlapping parts: $\Omega=\Omega_T\oplus\Omega_N$:

- Ω_T is the test set, in which all inputs to both versions have run successfully, i.e., $\theta_A(x)=0$ and $\theta_B(x)=0$ for all $x\in\Omega_T$;

- Ω_N comprises all inputs that were not tested, so the corresponding failure distributions can be assumed as $\theta_A(x)=\beta_A$ and $\theta_B(x)=\beta_B$ for all $x\in\Omega_N$. Similarly β_A and β_B are two constants, which describe the unknown parts as containing uniform failure distributions as in case 1. Then there is:

$$P_{AB} = \beta_A\beta_B \sum_{x\in\Omega_U} U(X=x) \geq P_A P_B = \beta_A\beta_B \sum_{x\in\Omega_U} U(X=x) \sum_{x\in\Omega_U} U(X=x) \quad (17)$$

This result shows that the independence model, in considering equal failure probability over the entire input space is optimistic for diversity estimation where the two versions are known with certainly to have some common successful input sets i.e. in nearly all realistic situations, due to testing.

This preliminary analysis has some implications, for example, suggesting that Statistical Testing of diverse software versions might benefit from use of different test sets for the different versions. Use of a common test set will reduce the randomness and therefore the theoretical diversity demonstration.

C. *Partition of Failure Distributions*

Equations (16) and (17) show that, in general, the IFP can not be used to estimate the CFP if the failure distribution is uneven over the whole input space. However, the independence condition may still exist on the input subsets when $\theta(x)$ is uniform over partition subsets. Based on the FD model: if the input space can be divided into N_C subspaces: $\Omega_i, i = 1..N_C$, and both $\theta_A(x)$ and $\theta_B(x)$ are constants on each subspace:

$$\theta_A(x \in \Omega_i) = \beta_A^i \quad (18)$$

$$\theta_B(x \in \Omega_i) = \beta_B^i \quad (19)$$

It is easy to prove:

$$\begin{aligned} P_{AB} &= P(\text{Both } \pi_A \text{ and } \pi_B \text{ fails on } X \in \Omega) \\ &= \sum_i^{N_i} (\beta_A^i \beta_B^i \sum_{x\in\Omega_i} U(X=x)) \end{aligned} \quad (20)$$

The equation (20) provides a simplified method to estimate diversity if the input space can be divided into parts with constant failure densities. The piecewise uniform distribution of failures is possible when there is limited number of failure patterns.

IV. CROSS VALIDATING THE FD MODEL USING AN EXAMPLE

A. *Background*

An online programming competition made it possible to collect a population of programs large enough to use the EL/LM models [21]. The MR experiment [22], used thousands of programs developed by different competitors to solve a single simple mathematical problem, and gives some statistical features of the faults and failures observed from the test results. The MR experiment studies only one type of program, which is not representative of safety related applications, and therefore cannot be used validate either the EL/LM models or the FD model as solutions to diversity modelling in this field. Although the general conclusions may not apply to real systems, the experiment was an important demonstration of the process of diversity model application, and it is important to compare the EL/LM and FD models in this respect.

The FD model assesses particular version pairs by evaluating each version's possible failure behaviors. A FI approach makes this measurement possible on any particular version pair, as illustrated on diversity assessment of a protection system [7]. An example in this section is constructed for the purposes of FD model and EL/ML model cross-validation, and demonstrating some advantages of the FD model in explaining diversity phenomena. The implementation required three steps:

1. A pair of programs Va and Vb is developed according to the common requirement specification for the "3n+1" mathematical issue in the MR experiment [22].
2. FI testing is carried out on the version pair:
   a. select, or provide a method of generating, representative faults (see below) for the two programs
   b. iterate the step of inserting a fault into a program and detecting its failure behaviors
   c. the stopping criteria of the iteration is exhaustive testing of all faults or when no new failure pattern appears. (This is a key potential source of uncertainty in the process, which will be discussed later in more detail)
3. The FD model (14) is used to calculate the diversity (reliability improvement).

The application of the FD model requires three parameters: operational profile, fault occurrence profile and the sign function for failure points in formula (11). To determine the former two subjective judgments are unavoidable in most realistic situations. The third parameter is found through FI tests. To assess Va and Vb and compare against the MR results, the parameters are determined as below:

- Possible faults and failure regions:

The number of possible faults for FI testing may be huge, but instead 'representative' faults can be used, based on an empirical observation in the MR experiment. Although thousands of different faults were detected in the programs, only 20 failure patterns were identified (with labels as *a* to *t* by figure 1 in [22]). Similarly, the MR experiment categorized all the detected faults into 37 equivalence classes (with labels as EC0 to EC36 in table 1 in the appendix). Any fault injected from the MR fault set into Va or Vb will cause one of the 20 patterns, and any fault belonging to one of the 37 categories can be mapped to one of the 20 patterns. A premise of the MR experiment is that the fault set is in some way representative or adequate, and we will use that same premise. The $\nu(f,x)$ in (11) can be obtained based on the failure patterns of the versions.

- Fault occurrence frequency and operational profile.

These two parameters $H(F=f)$ and $U(X=x)$ can be obtained from the MR data: the MR experiment operation profile can be used and the frequency of representative faults and propagation of the fault-failure can be obtained from table 1 in the appendix. The determination of operational profiles has been discussed in applications of statistical models for failure probability estimation [26]. The fault frequency is empirical in this MR example, but in general will not be available, and the assessment will have to be made on the basis of an agreed assumption about how to distribute fault frequencies for the purposes of assessment. However, for the purposes of comparison with MR results, the fault frequency can be calculated from the MR program population.

The above discussion shows that the data required by FD models to assess the Va and Vb pair can all be deduced from the MR results. Therefore this comparison of the FD and EL/ML models is based on the consideration of a virtual version pair Va and Vb, containing all the types faults (or failure patterns) found in the MR experiments. Estimation for a particular real version pair would only differ in the selection of representative faults and their occurrence frequencies.

B. Comparisons of the models

Cross validation of the models proceeds by comparing the results regarding reliability improvement (RI). For example, a RI index for the 1-out-of-2 system may be $\min\{P_A, P_B\}/P_{AB}$, where $\min\{P_A, P_B\}$ is the smaller of P_A and P_B [4].

One of the main observations in the MR experiment concerned the influence of fault sizes on software diversity. A single common large population of programs was defined for the selection of a version pair, so P_A and P_B have the same value: the average failure rate (AFR) over the version population. The RI is AFR/P_{AB} in this case. Correspondingly for the cross-comparison, the FD model uses the AFR of the version over the fault set. The procedure in the MR experiment was to remove the most unreliable program from the program pool each time to reduce the AFR of the program population, then to observe the change of RI. Accordingly a FD experiment simulates the process by removing faults in the fault set progressively according to their size from big to small (in order of the failure rate i.e. from top to bottom in the first column of table 1 in the appendix). This generates the results in fig. 2.

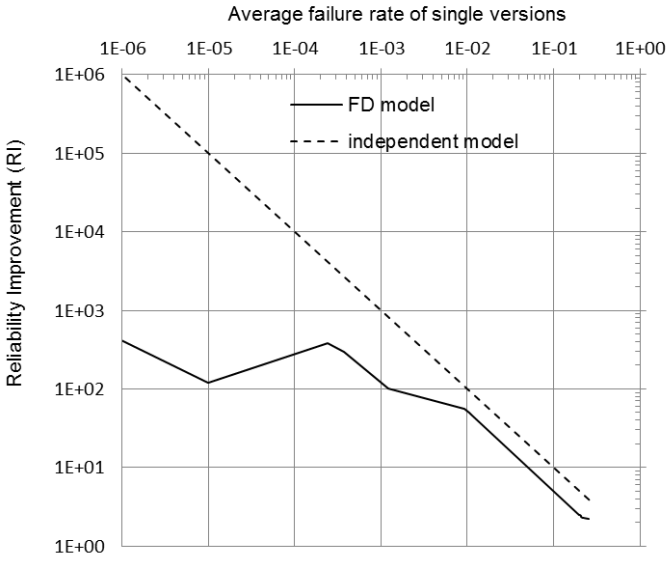


Fig. 2. Reliability improvement for 1-out-of-2 pair (FD model)

Two RI to AFR relations are displayed in the fig. 2:

1. the dashed line shows what happens when common failure probability obeys the IFP model, and is a straight line (RI is the inverse of AFR), included for purposes of comparison.
2. the solid line shows results from the FD model. The result essentially repeats that in the MR experiment [22], but with unnoticeable differences because of using 4943 (rather than 5897 originally used) as the total number of programs to get the *P(F)* in table1 of the appendix. This different number is obtained by re-adding up the different programs according to the data introduced in the MR paper.

This example uses the MR results to replace the need for fault injection. The aim was to use the MR results to show how the FD model explains diversity behaviors in a different way. The source data required by the FD model is not in an identical format to that in the MR experiment, so the above results primarily validate the data regenerated from the MR experimental results for use in FD model. The FD model produced an equivalent diversity assessment to the EL/LM model from a same testing process, providing a simple cross validation of the two models. However, the FD model can also be used to produce different results from those in the MR experiment, enhancing insight into diversity phenomena. Concerning the RI-AFR relation, one observation from the MR results is that pairs comprising lower reliability versions tend to show a RI closer to the IFP model (solid line is closer to the dash line for higher AFR in the figure 2, producing a more desirable RI). This trend was explained as being dominated by fault sizes. However the FD model can be used to show that that reliability improvement by diversity in particular version pairs is fundamentally influenced by failure patterns and does not have a simple relationship with SFP (i.e. AFR, because the average failure probability of the population was used to represent the SFP). A special case was designed for this purpose: the failure patterns were removed one by one from bottom to top according to the sequence in the first column of table 1 in appendix (from small faults to big faults). The corresponding result is shown in fig. 3.

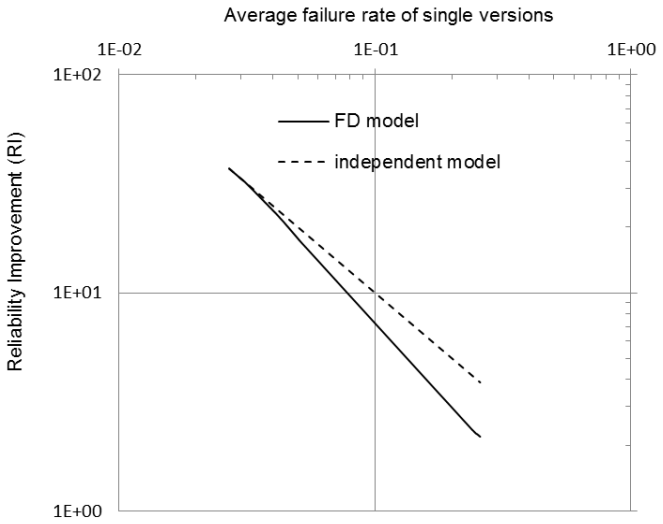


Fig. 3 Influence of failure patterns on reliability improvement

In contrast to the MR-experiment results in fig. 2, it can be seen that a pair comprising more reliable programs is closer to the IFP model in terms of RI. The best pair can even achieve a level of diversity equivalent to that produced by the independence model.

This simple study of the same program pool (or fault set) demonstrates that the average SFP of the pool is not the essential factor influencing reliability improvement by diversity. The phenomenon in figure 3 illustrates the influence of the failure patterns on diversity. In (17), the diversity is mainly decided by the size of the possible area for failure distribution in an input space. Under the simplified conditions, the ratio between CFP and IFP is $P_{AB} / P_A P_B = \sum_{x \in \Omega_U} P(X = x)$. In fig. 3, the average SFP decreased as faults were removed one by one, but the possible failure areas remained as the whole input space (because the failure pattern (f) with 100% failure rate is included). The density of failure distribution was tending to uniform and $\sum_{x \in \Omega_U} P(X = x) = 1$ for the manipulated pool after the final removal, which resulted in the diversity measure fitting the independence model.

V. DIFFERENCES BETWEEN THE MODELS

A. Setup of a new test

Comparing FD and EL/LM models using common MR test results showed essentially the same results but some differences concerning the explanation of diversity. Further to the MR experiment, a New test was constructed to demonstrate some distinct abilities of the FD model for estimation of diversity in particular Version-Pairs (NVP test). The NVP test aims to illustrate the process of applying the FD model and fault injection of 'small' faults (a feature of safety-critical systems), and to compare analysis of a particular version pair (FD) with analysis of a large population of programs (EL/LM).

The NVP test is concerned with two main observations from the MR experiment:

1. most failure patterns were caused by big faults, and 14 out of 20 have a failure rates higher than 40%, which corresponds to 30 out of the total 37 classes of faults (including the zero size faults). These big faults have a major influence on the statistical analysis.
2. the experiment used only 2500 demands/inputs so the test results did not reveal realistic failure behaviors (i.e. for small faults, for example, where the SFP or AFR is below 10^{-4}). The low SFPs (<1/2500) in both the MR and FD experiments were results of averaging over a large population of programs with high variation (including fault-free and totally-failed programs).

The NVP test examines whether these conclusions are relevant to real software with high reliability, by articulating further the roles of failure patterns and fault sizes. It reuses as far as possible the test resources in the MR experiment:

- A version pair for FI test is constructed from two copies of the standard C version provided in the paper [22]. A calculation of diversities of the two copies (containing different faults) is used to simulate a version pair programed by two different developers based on the same software structure chart. A similar observation was made in the MR experiment: many competitors submitted the same or very similar programs to the standard C version.
- In addition to the exhaustive test of the original input space of size 50X50 (SP1), a new 100X100 space (SP2) is setup. The new input space includes the original one: the range of the two input parameters are changed from [1, 50] to [1,100]. This extension was made in order to be able to track real small faults with failure probabilities between $10^{-4} \sim 10^{-3}$, whilst maintaining comparability of the faults and consequences from NVP FI tests with the original classifications in the MR experiment.

B. Implementation of FI

The aim of FI in the context of the FD model is to find all failure patterns. This involves a conjecture that the FI is sufficiently powerful. This is equivalent to the conjecture in the EL/LM approach that the population of developed programs is sufficiently complete and varied. Once failure patterns are established, it is possible to experiment with different pattern occurrence frequencies. As in the published EL/LM approach, there is no assurance that any set of frequencies is 'valid' or 'representative of reality', but in the FD approach the ability to vary frequencies provides rich information about the natural propensity of two specific program versions to exhibit diverse failure behaviours.

Both representative and exhaustive fault insertion strategies are used in the NVP test:

- Representative injection is based on partitioning the program structure chart. This reverses the process of fault classification in the MR experiments where the faults were labeled according the functional part (component) they occurred in. Four components and corresponding fault types in the C version were defined as: *Swap, Loop, Calculation* and *Output*. The NVP test keeps this partition. Although other approaches could be applied, the reuse of the partition based on functional division with the corresponding frequencies makes for convenient comparison between the NVP and the MR results.
- The 'exhaustive' injections attempt to simulate the widest range of faults possible, in order to reveal the failure distribution over the input space. The number of potential faults may be infinite, so "no new failure pattern appears" is used as a FI stop criteria. The FI operation follows a top-down procedure based on the program structure: creating anomalies in the I/O of the functional components from high to low levels of abstraction (big to small components).

Fault injection in the NVP is achieved by code modification, i.e. replacing the original code with distorted code. The degree to which code is alterable is constrained by the compilation process. The standard C version uses concise coding. So, although C admits high flexibility in programming, nevertheless the opportunities to insert faults is limited. Therefore in this case, exhaustive FI becomes achievable if only appropriate faults are counted in. For example, fault EC22 in the MR experiment in table 1 is the omission of the first value in the loop:

- : *for (i = min(a, b) + 1; i <= max(a, b); i++),*

where the variable *i* should start directly from *a* or *b, i.e.* the correct code is *'i = min(a, b)'*. The fault injection to simulate the EC22 type of faults might also be e.g. '*i = min(a, b) + 2'* or *'+3'* etc.. All other possible faults at this injection point includes any perturbation of the variable *i* by value alterations using various numerical operators, or replace the variables *i, a* and *b* with different ones. Even so, the number of possible simulations is not very high because of: 1) the control of the compiling process; 2) that larger perturbations of the *min(a,b)* increase the failure probability, so the requirement to use small faults means that most faults of this type are inadmissible.

The NVP test eliminated the faults with zero and 100% failure probability, and those faults that failed the compiling process to restrict attention to realistic fault conditions for software diversity estimation.

The NVP experiment exhaustively tested the program with an inserted fault on both input spaces, i.e., the failure behaviours of each fault are compared on SP1 and SP2. The list of injected faults can be provided by the authors if required.

C. Observations and results

The extended test space provides some understanding of the influence of different input space definitions on diversity estimation. Two contrary fault-failure relations are observed:

1. Some faults preserved the same failure patterns and similar failure probabilities in the two different input spaces SP1 and SP2. These were faults in the decision component of the flowchart that is the *swap* functional part in the C version. Table 2 shows some of the faults with failure patterns also contained in table1.

   TABLE 2 SOME FAULTS WITH SAME FAILURE PATTERNS ON THE TWO INPUT SPACES

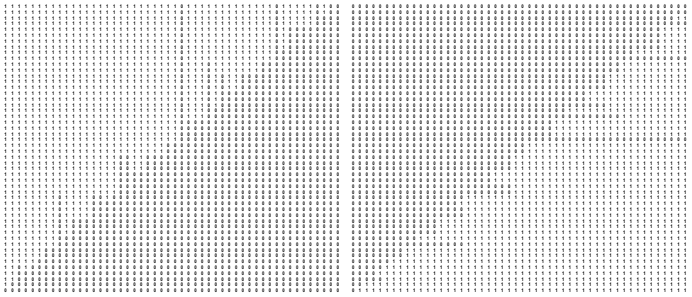
Fault	(l): EC18	(c): EC04	(b): EC01 EC02 EC03 EC05 EC15
P on SP1	45.04%	42%	49%
P on SP2	46.76%	43.87%	49.5%

   The algorithm used has symmetry on the two input parameters and the faults in the swap component normally impact half the input space. In the MR experiment, all *swap* faults, including table 2, have a big size, are found in 58% of 3414 faulty C language. Versions, and 59% of 8057 faulty versions in all three different languages: C, C++ and Pascal. The FI process observed a similar ratio: more than half of feasibly injected faults are in the *swap* component.

2. Other faults changed their failure probability in proportion to the sizes of the input spaces. This usually happened at a part of the boundary of a subspace corresponding to a sub-function of the program. One example is the *loop* fault EC22 that skipped the first input. For both SP1 and SP2, this fault only caused one failure point (the failure pattern (m) of figure 1 in [22]), but has failure probabilities 0.04% and 0.01% respectively. Only one such type of fault can be simulated in the FI, which responds to the observations in MR experiment: it rarely happened, in just 0.26% of 3414 C versions and 0.45% of 8057 all versions.

The two situations indicate that the definition of input spaces sometimes influences the failure probabilities of the failure patterns, and therefore the diversity estimation by the FD model needs to specify the input space. These phenomena differ with the conclusions in the corresponding part of the MR experiment that compared three test regimes: complete test of the SP1, random test of SP2, and complete test of a part of the SP2, and found no differences with respect to the findings in its result analyses related to the choice of test regime or number of inputs.

The FD model emphasizes the dominance of the failure patterns in calculation of correlation among versions. The FI in NVP test aims at an exhaustive search of failure patterns, but didn't expect to reveal many more new patterns than the MR experiment. The FI process did detect some new patterns, but not all the existing 20 patterns in the table2. The unobserved patterns (n), (j), (o), (p), (e), (k), (t), (i) and (g) represent 11 of the 37 fault types, and are comprehended as only occurring in some different programming structures. The new patterns were practically unlikely faults, e.g., neglected or miscalculated the first few inputs (rather than only the first one as in EC22) in *loop*. A new finding is that the failure patterns caused by faults in *swap* normally appear as symmetric pairs to the two input parameters constructing the input space. Figure 4 gives an example of the pattern pair in SP2, where a light-grey dot is 1 (failure input) and a dark-grey dot is 0 (success input), with a compressed font-size. The FR denoting failure rate is an average value of all elements in a matrix. In figure 4, pattern (x) corresponds to MR pattern (l) for EC18 in the table 1 and pattern (y) is a newly identified one. The symmetric patterns for (b), (c), (d) and (q) are also detected in the FI test as supplements to figure 1 in [22].



(x) (y)

Fig. 4 A symmetry pair of failure patterns in SP2

The exhaustive FI simulated 105 faults that could pass compilation and had failure probabilities between 0 and 1 (exclusive). Tables 3 and 4 summarize the results for test on SP1 and SP2 respectively. The uniform usage profile has been used for RI calculation, as in figure 2. The uniform profile is also applied to each of the 105 injected faults, which approximately matches the profile used in the figure 2.

TABLE 3 FI RESULTS OF THE C VERSION PAIR ON SP1

Fault Cat.	$<10^{-0}$	$<10^{-1}$	$<10^{-2}$	$<10^{-3}$
Fault Amount	105	34	8	2
Ave. Failure Rate	0.371554	0.047282	0.001700	0.000400
RI	2.553129	3.520171	1.373737	1.000000

TABLE 4 FI RESULTS OF THE C VERSION PAIR ON SP2

Fault Cat.	$<10^{-0}$	$<10^{-1}$	$<10^{-2}$	$<10^{-3}$
Fault Amount	105	44	9	8
Ave. Failure Rate	0.353817	0.043168	0.001078	0.000425
RI	2.714826	3.727120	2.653495	1.373737

In tables 3&4, the first row categorizes faults by their sizes. The following rows give the number of faults and their average failure rate, and the reliability improvement under each category. For example, the fifth column in table 3 is for a diversity estimation of the version pair with SFP less than 10^{-3} in SP1. It indicates that the exhaustive FI detected only two faults with an average failure rate 0.04% and there is no diversity (by RI=1) for this 1-out-of-2 system under the given input space definition and the specified SFP. A more general result is given in figure 5.

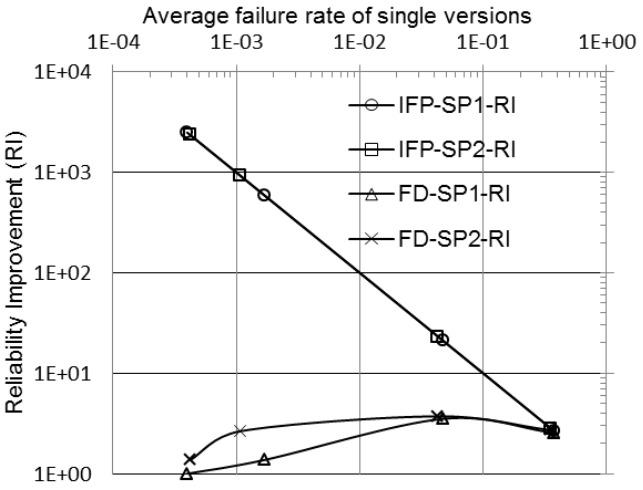


Fig.5 Diversity estimation of the C-version pair by FD model

Four RI-SFP relations are shown in figure 5: the IFP-SP1-RI and IFP-SP2-RI displayed as the duplicated diagonal lines give a same estimation based on the IFP model in the two input spaces; the FP-SP1-RI and FD-SP2-RI provide measurements based on the FD model in SP1&2 respectively. These results describe some diversity features of the 1-out-of-2 system comprising the standard C version pair:

- The definition of input space may influence the diversity estimation. For this particular version pair, the differences occur at low SFP because the SP2 definition increases the ratio of small faults and adds new failure patterns. Tables 3&4 show that fault numbers change from 34 to 44 for sizes smaller than 10% when the SP1 is enlarged four times to give SP2.
- The increase of small faults and their patterns extend the region where the failures by small faults locate. The FD model indicates that this extension of failure distribution area makes the diversity in SP2 bigger than the one in SP1 when the SFP of the version pair is between 10^{-4} to 10^{-1}.
- The FI test demonstrates that the opportunity for the NVP to contain small faults is very rare, a point also observed in the MR experiment. This makes the diversity of the version pair very low. An extreme case is the 'zero diversity' or no reliability improvement' when RI=1 for SFP less than 10^{-3} in SP1, which happens because it is only possible to inject two faults of appropriate size, and more importantly these two faults have the same failure pattern.
- Comparing the FD-RI curves to the IFP-RI ones, overall the diversity of the version pair estimated by the FD is much lower than the IFP model produces. This is true over the whole range of SFPs, and the diversity is even obviously lower than the average diversity of the thousands of programs in figure 2. The reason for the difference between the two estimations based on EL/LM model and FD model is due to the influence of failure distribution in the models as explained in figure 3 based on formula (17).

VI. DISCUSSION

Both EL/LM and FD models study diversity based on the consideration of statistically averaged failure behaviors:

- EL/LM considers all possible programs designed to a common requirement specification.
- FD considers a particular program-pair under all potential faulty conditions.

It has been shown how the FD model can be described using the theoretical foundations developed for EL/LM models, but is more specific to particular program pairs. The FD model appears to offer potential for development into a practical diversity measurement method. As indicated in section II.D, the collection of failure behaviors is shifted into the FI implementation in the FD model from the program generation in the EL/LM model. There are two further barriers to obtaining useful program populations for the EL/LM model, especially on safety-critical systems. Firstly, the high cost to produce diverse software versions using different functions, languages and development teams. Secondly, the low incidence of software failures in high integrity software. Failures are expected to be very rare in such software prior to operation (because any revealed faults are removed). Yet a practical, general diversity evaluation method based on the EL/LM model relies on the presence of faults in the program versions.

Some recent progress on applications of diversity to enhance quality attributes of a system does promote the promising future for techniques that generate a large population of programs. For example, an approach synthesizing program variants automatically has been developed and proposed to increase system security [33]. At a given time, this technique randomly selects one program from a population of computationally diverse program variants (Sosies) to make it hard for hackers to focus on program features to attack. This kind of automatic programming technique makes the production of a large number of programs with a common requirement specification more affordable. However, some essential difficulties need to be overcome for its application:

- Functionally diverse program variants are more difficult to generate than the Sosies. Like the data diversity used as a fault-tolerant method in safety-critical systems, Sosies only provide effective defence against some specific issues [34, 35].
- The Sosies don't show diverse behaviors (i.e they contain no faults other than those present in the original version) under present conditions, which makes it impossible to use the EL/LM model. It might be possible to introduce automatic fault seeding into Sosies using Mutation [13], but this is beyond the currently published work and is an approach that would require the FD model to evaluate the diversity benefits.

In practical application of the FD model based on fault injection, the major difficulties will be the selection of all possible faulty conditions and the fault occurrence frequencies. However, the empirical observations of limited failure patterns may help to reduce the burden, allowing the use of representative faults rather than all them. One approach to constructing representative faults uses simulated anomalies at all branch paths of data and control flows that influence where failures occur in an input space. The selection of the paths is a top-down approach: from the linkages of big functional units to small ones.

This work has focused on cross-validation of the FD model with existing work and its use to explain diversity phenomena. However, to achieve practical use of the FD model will require further research questions to be overcome. For example, the proposed FI criterion to stop inserting faults is that no new failure patterns would be revealed under the simulated faulty condition. However, there is no way to know this for sure and the possibility of neglecting some failure patterns in FI testing can not be excluded. Uncertainty analyses can help to estimate and confine its influence on diversity assessment, e.g., assuming representativeness of selected faults, the weight of any neglected failure patterns in the profile of faulty condition is calculable reduces as the number of tested faults increases; also test set construction plus stress tests can be used to check if the failure patterns cover branches of data/control flow and the entire input space. Nevertheless, the systematic identification of failure patterns remains an open research question.

It has been shown that a high variety of failure patterns is a key benefit in diversity design, as shown mathematically in III.B and illustrated empirically in IV.B. Potentially in practical design, it can be used to enhance diversity through choice/adjustment of software structure. However, this must be balanced against the possibility that complex structures raise the risk of introducing more faults. The balance between these two phenomena needs to be better understood.

The NVP experiment investigated the problem of estimating diversity of a particular program pair, its complexities and distinctions from the concept of an average common failure probability of a large population of possible versions based on the EL/LM model. For the "3n+1" algorithm and the corresponding program in the C language, the failure patterns are very limited, especially for the small faults. So for such two versions potentially containing small faults, the expectation for the pair to show different failure behavior is very low. Most failure patterns in the MR experiment were caused by big faults. The calculation of SFPs by averaging over a large population of programs, including perfect versions, assumed implicitly a bigger than real sub input space for the potential small faults to distribute their failures. This overstated possible divergence in failure behavior when the versions contain small faults.

In the application of fault injection to assessment of software diversity, sieving of faults based on (failure probability) size can be used to study software with high reliability requirements. The main threat to validity of the assessment based on the FD model is the estimation of the parameters needed by the model:

- identification of the failure patterns,
- estimation of occurrence frequencies of the simulated faults or failure patterns,
- estimation of the operational profile of inputs

All three factors may affect the results. The first factor is determined by the completeness of the method of fault injection, and remains an open problem. It is similar to the problem of whether a population of programs, created by different people for assessment using the EL/LM models, is complete in some sense. The latter two of the three factors are similar to the uncertainties in all other Statistical Testing models, and may require expert judgments. One potentially promising way forward would be to study the sensitivity of the diversity assessment to these uncertainties. In the NVP test, the variety of potential faults in a functional component was compatible with the fault frequency in the programs collected by the MR experiment, and the fault variety for FI was related to the complexity and LOC (lines of code) of the functional component.

VII. Conclusion and Future work

A model to assess software diversity based on fault injection test has been constructed and compared against existing models and a diversity experiment in the literature. The new model has been used to explain the influence of failure distributions on diversity. Furthermore, an experiment has been conducted demonstrating the distinction between the FD model and previous similar models in the literature, in particular its ability to study diversity estimation of a particular version pair. It is a potential future basis for a new form of safety argument to be used in software safety cases.

To use the model in a safety case it would be necessary to devise a different form of argument to those that are accepted currently. The argument is that if two devices (or software versions) show diversity for all the possible failure patterns, it will show diversity with a high probability for any random set of real faults under the same conditions. If convincing evidence for determining potential faults and fault occurrence frequencies can be determined to support this argument, it will become reasonable to make a diversity claim as part of safety engineering (and hence claim improvement in the reliability of a multi-version system over its constituent single versions). Further use of experience in other mutation test applications is required to find a practical approach to judge the representativeness and the adequacy of fault selections. The applicability of the FD model will also depend on the application problem e.g. some special diversity designs, such as those designed to defend against specific common cause failures, will make it easier to specify faulty conditions. Future work should include sensitivity analysis to bound reliability improvement claims due to the uncertain factors in the FD application process.

Use of fault injection concepts could be extended to assess other attributes of a system with diversity e.g. analysis of the security of a system using Sosies. In that scenario, fault injection translates into simulated attacks, and the FD model would be used to estimate the ability of a Sosie-based system architecture to defend against them.

Acknowledgement and Disclaimer:

The work presented in this paper comprise aspects of a study (NewDISPO) performend as part of UK Nuclear Safety Research programme, funded and controlled by the C&I Nuclear Industry Forum (CINIF), EDF Energy and the Health and Safety Executive. The views expressed in this paper are those of the author(s) and do not necessarily represent the views of the members of the C&I Nuclear Industry Forum (CINIF). CINIF does not accept liability for any damage or loss incurred as a result of the information contained in this paper.

References

[1] A. Avizienis, "The N-version approach to fault-tolerant software," *IEEE Trans. Softw. Eng*., vol.11, no. 12, pp. 1491-1501,1985.

[2] D. Brire and P. Traverse, "Airbus A320/A330/A340 electrical flight controls - a family of fault-tolerant systems," in *Proc. of 23rd Int. Symp. on Fault-Tolerant Computing,* Toulouse, France, 1993, pp.616-623.

[3] G. Hagelin, "Ericsson safety system for railway control," in *Software Diversity in Computerised Control Systems*, U. Voges, Vienna, Springer-Verlag, pp. 11-22, 1988.

[4] A.E. Condor and G.J. Hinton, "Fault tolerant and fail-safe design of CANDU computerized shutdown systems," *IAEA Bulletin*, vol. 27(3) pp. 7-12, 1985.

[5] M.R. Lyu(ED), "Handbook of software reliability engineering," *IEEE Comp. Society Press,* 1996.

[6] J.McDermid and T. Kelly, "Software in safety critical systems: achievement and prediction," *Nuclear Future*, vol 2, no. 3, pp. 6,2006.

[7] J.M. Voas and G. McGraw, "Software fault injection: inoculating programs against errors," *Wiley Computer Publishing*, 1998

[8] L. Chen, J. May and G. Hughes, "Estimation of software diversity by fault simulation and failure searching," in *Proc. 12th ISSRE*, Hong Kong, 2001, pp. 122-131.

[9] G.F. Carpenter, "Mechanism for evaluating the effectiveness of software fault-tolerant structures," *Microprocessors and Microsystems*, vol. 14(8), pp.505-510, 1990.

[10] D. Avresky, J. Arlat, J. C. Laprie and Y. Crouzet, "Fault injection for formal testing of fault tolerance," *IEEE Trans. Rel.*, vol.45, no.3, pp.443-455, 1996.

[11] J. Napier, L. Chen, J. May and G. Hughes, "Fault simulation to validate fault-tolerance in Ada," *Int. J. of Comp. Sys.: Sci. & Eng.,* vol. 15, pp. 61-67, 2000.

[12] J. A. Clark and D. K. Pradham, "Fault Injection, a method for validating computer-system dependability," *IEEE Comp.*, June, pp.47 – 56, 1995.

[13] R. A. DeMillo, D. S. Guind, W. M. McCracken, A. J. Offutt and K. N. King, "An extended overview of the Mothra software testing environment," in *Proc. of Second Workshop on Software Testing, Verification, and Analysis*, Banff, Canada, July 1988, pp.142-151.

[14] J.C. Knight and N.G. Leveson, "An experimental evaluation of the assumption of independence in multiversion programming," *IEEE Tran. Softw. Eng.,* vol.12, no.1, pp.96-109, 1986.

[15] D.E. Eckhardt, A.K. Caglayan, J.C. Knight, L.D. Lee, D.F. McAllister, M.A. Vouk and J.P.J. Kelly, "An experimental evaluation of software redundancy as a strategy for improving reliability," *IEEE Trans. Softw. Eng.*, vol.17, no.7, pp. 692–702, 1991.

[16] A. Avizienis, M.R. Lyu and W. Schuetz, "In search of effective diversity: a six language study of fault tolerant flight control software," In *Proc. FTCS-18*, Tokyo, 1988, pp.15-22.

[17] J.P.J. Kelly, D.E. Eckhardt, M.A. Vouk, D.F. McAllister, and A. Caglayan, "A large scale second generation experiment in multi-version software: description and early results,". In *Proc. FTCS-18*, Tokyo, 1988, pp.9-14.

[18] M.R. Lyu and Y. He, "Improving the N-version programming process through the evolution of a design paradigm," *IEEE Trans. Rel*., vol.42, no.2, pp.179–189, 1993.

[19] D. Eckhardt and L. Lee, "A theoretical basis for the analysis of multiversion software subject to coincident errors," *IEEE Trans. Softw. Eng.*, vol. 11, no.12, pp.1511-1517,1985.

[20] B. Littlewood and D. Miller, "Conceptual modeling of coincident failures in multiversion software," *IEEE Trans. Softw. Eng.*, vol.15, no.12, pp.1596-1614, 1985.

[21] M. van der Meulen, P. G. Bishop and R. Villa, "An exploration of software faults and failure behaviour in a large population of programs," in *Proc. ISSRE-15*, 2004, pp.101-112

[22] M. van der Meulen, and R. Villa, "The effectiveness of software diversity in a large population of programs," *IEEE Trans. Softw. Eng.*, vol. 34, no. 6, pp. 753-764, 2008.

[23] IEEE Systems and software engineering - Vocabulary*, ISO/IEC/IEEE 24765*, 2010

[25] J. Knight and N. Leveson, "An experimental evaluation of the assumption of independence in multi-version programming," *IEEE Trans. Softw. Eng.,* vol.12, no.1, pp.96-10,1986.

[26] J.D. Musa, "Operational profiles in software reliability engineering," *IEEE Software*, vol. 10, pp.14-32, March, 1993.

[27] J. L. Rodgers, W. A. Nicewander and L. Toothaker, "Linearly independent, orthogonal, and uncorrelated variables," *The American Statistician*, vol. 38, no. 2. pp. 133-134, 1984.

[28] P.G. Bishop, "The variation of software survival time for different operational input profiles," in *Proc. 23rd Int. Symp. on Fault-Tolerant Computing*, Toulouse, France, 1993, pp.98-107.

[29] L. Chen and J. May, "Safety assessment of systems embedded with COTS components by PIP technique," in *Proc. SOQUA/TECOS 2004*, Erfurt, Germany, 2004, pp.93-108

[30] P.G. Bishop and R.E. Bloomfield, "A methodology for safety case development," in *Proc. Safety-Critical Systems Symposium*, Birmingham, UK, February 1998, pp.194-203

[31] R. Weaver, J. Fenn and T. Kelly, "A pragmatic approach to reasoning about the assurance of safety arguments," in *Proc. 8th SCS*, Darlinghurst, Australia, 2003, pp. 57-67.

[32] International Electrotechnical Commission, "Function Safety of Electrical/Electronic/Programmable Safety-Related Systems, Parts 1-7," *IEC 61508*, various dates.

[33] B. Baudry, S. Allier and M. Monperrus, "Tailored source code transformations to synthesize computationally diverse program variants" in *Proc. ISSTA-2014*, San Jose, CA, USA, 2014, pp. 149-159.

[34] E. Paule, J. Ammann and C. Knight, "Data diversity: An approach to software fault tolerance," *IEEE Trans. Compu.*, Vol.37, No.4, pp.4188-425,1988.

[35] D.K. Pradhan, "Fault-Tolerant Computer System Design," *Computer Science Press*, 2003.

[36] B.Littlewood and A. Povyakalo, "A conservative reasoning about the probability of failure on demand of a 1-out-of-2 software-based system in which one channel is "possibly perfect"," *IEEE Trans. Softw. Eng.*, vol.39. no.11, pp. 1521-1530, 2013.

[37] P. Bishop, R. Bloomfield, B. Littlewood, P. Popov, A. Povyakal and L.Strigini, "A conservative bound for the probability of failure of a 1-out-of-2 protection system with one hardware-only and one software-based protection train," *Reliability Engineering & System Safety*, vol.130,pp. 61-68, 2014.

Luping Chen obtained his Ph.D. from the State Key Laboratory of Structural Analysis for Industrial Equipment, Dalian University of Technology for work on model reduction and optimization of large-scale and complex control systems. He is a CINIF Researcher in Safety System Research Centre of the University of Bristol, and focuses on design and test of fault defence mechanisms in control systems, statistical test of protection systems and fast test techniques. His current interests include digital system reliability and safety, software design for test.

John H. R. May received the Engineering Mathematics BSc degree in 1983 and a PhD degree (Engineering Safety) in 1986 from the University of Bristol, UK. He has worked at the University of Southampton, the Open University, and the University of Bristol, where he holds the post of Reader in Safety Systems. His current research interests are in reliability assurance of digital systems, with a focus on programmable systems and wireless networks, and organisational resilience to the causes of disasters.

APPENDIX DATA REGENERATED FOR FD MODEL BASED ON THE MR EXPERIMENT

From the published MR experimental information, the 20 failure patterns from *a* to *t* on the 50X50 point matrix (input space) which is the same as displayed by figure 1 in [22], and corresponding failure rates (FR) can be regenerated for the FD calculation. The mapping relations of the fault-types to the fault-patterns identified by MR experiment are listed as the third column in table 1. The 20 failure patterns for all 37 classes of faults are covered by the 20 1-0 matrices (with the same format of figure 4). In the MR experiment, the occurrences of all 37 faults from EC00 to EC 36 in the thousands programs have been recorded. This information can be converted to give frequencies for each fault, which can be summed to give frequencies for each failure pattern (the fourth column in table 1)

TABLE 1 RELATION BETWEEN FAULTS AND FAILURES (C LANGUAGE VERSION)

Failure pattern	Failure rate	Faults linked to the failure pattern		SumP(F)[i]
f	1	EC08 EC11 EC12 EC17 EC21 EC25 EC27 EC29 EC31		0.026907
s	0.9996	EC33		0.000809
n	0.7736	EC23		0.002428
j	0.5976	EC14		0.003237
d	0.5624	EC06		0.017196
o	0.562	EC24		0.002428
q	0.54	EC30		0.001416
p	0.5168	EC28		0.001012
e	0.4908	EC07 EC19 EC20		0.012543
k	0.4904	EC16		0.001821
b	0.49	EC01 EC02 EC03 EC05 EC15 EC34[ii]		0.384989
t	0.4724	EC36		0.001214
l	0.4504	EC18		0.00263
c	0.42	EC04		0.019219
h	0.1248	EC10		0.006676
r	0.1244	EC32		0.001012
i	0.1048	EC13		0.002225
g	0.0012	EC09		0.007485
m	0.0004	EC22 EC35		0.002428
a	0	EC00		0.502327

[1] The last row in MR-table gives a total number of programs by C language as 5897. The actual sum from the table in MR paper is 4943 and used here for this calculation.

[2] The relation of fault EC34 to failure pattern was not found in MR-figure 1 as others. This mapping is a speculated identification based on the unique reliability figure of each failure pattern.